Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Отчет по лабораторной работе №8

«Исследование потоковых шифров»

Студентка: Пунько А.А,

ФИТ 3 курс 5 группа

Преподаватель: Берников В. О.

Минск 2020

**Цель:** изучение и приобретение практических навыков разработки и использования приложений для реализации потоковых шифров.

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости потоковых шифров.
2. Разработать приложение для реализации указанных преподавателем методов генерации ключевой информации и ее использования для потокового зашифрования/расшифрования.
3. Выполнить анализ криптостойкости потоковых шифров.
4. Оценить скорость зашифрования/расшифрования реализованных шифров.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

**Теоретические сведения**

Потоковый шифр (иногда говорят «поточный») – симметричный шифр, преобразующий каждый символ mi открытого текста в символ шифрованного, ci, зависящий от ключа и расположения символа в тексте. Термин «потоковый шифр» обычно используется в том случае, когда шифруемые символы открытого текста представляются одной буквой, битом или реже – байтом. Все потоковые шифры делятся на 2 класса: синхронные и асинхронные (или самосинхронизирующиеся). Основной задачей потоковых шифров является выработка некоторой последовательности (гаммы) для зашифрования, т.е. выходная гамма является ключевым потоком (ключом) для сообщения

Синхронные потоковые шифры (СПШ) характеризуются тем поток ключей генерируется независимо от открытого текста и шифртекста. Главное свойство СПШ – нераспространение ошибок. Ошибки отсутствуют, пока работают синхронно шифровальное и дешифровальное устройства отправителя и получателя информации. Один из методов борьбы с рассинхронизацией – разбить отрытый текст на отрезки, начало и конец которых выделить вставкой контрольных меток (специальных маркеров).

Синхронные потоковые шифры уязвимы к атакам на основе изменения отдельных бит шифртекста. В самосинхронизирующихся потоковых шифрах символы ключевой гаммы зависят от исходного секретного ключа шифра и от конечного числа последних знаков зашифрованного текста. Основная идея заключается в том, что внутреннее состояние генератора потока ключей является функцией фиксированного числа предыдущих битов шифртекста. Поэтому генератор потока ключей на приемной стороне, приняв фиксированное число битов, автоматически синхронизируется с генератором гаммы. Недостаток этих потоковых шифров – распространение ошибок, так как искажение одного бита в процессе передачи шифртекста приведет к искажению нескольких битов гаммы и, соответственно, расшифрованного сообщения.

Часто используемый алгоритм генерирования (программно или аппаратно) ПСП реализуется на основе так называемого линейного конгруэнтного генератора, описываемого следующим рекуррентным соотношением:

xt+1 = (a\*xt + c) mod n,

где: xt и xt+1 – соответственно t-й (предыдущий) и (t+1)-й (текущий, вычисляемый) члены числовой последовательности; а, с и n – константы. Период такого генератора (период ПСП) не превышает n. Если параметры a, b и c выбраны правильно, то генератор будет порождать случайные числа с максимальным периодом, равным c. При программной реализации значение с обычно устанавливается равным 2b-1 или 2b, где b – длина слова в битах. Достоинством линейных конгруэнтных генераторов псевдослучайных чисел является их простота и высокая скорость получения псевдослучайных значений. Линейные конгруэнтные генераторы находят применение при решении задач моделирования и математической статистики, однако в криптографических целях их нельзя рекомендовать к использованию, так как специалисты по криптоанализу научились восстанавливать всю последовательность ПСЧ по нескольким ее значениям. Генератор практически не используются в криптографии в силу низкой криптостойкости. Тем не менее, полезны для решения задач моделирования. Комбинации нескольких (чаще двух) линейных конгруэнтных генераторов позволяют значительно повысить период ПСП. Б. Шнайер, например, приводит данные о том, как на 32-разрядных ПК реализовать генератор в виде комбинации двух, каждый из которых обеспечивает период соответственно 231 – 85 и 231 – 249, а комбинированный генератор позволяет достичь периода ПСП, равного произведению указанных чисел.

Алгоритм RC4 разработан Р. Ривестом в 1987 г. Представляет собой потоковый шифр с переменным размером ключа. Алгоритм RC4, как и любой потоковый шифр, строится на основе генератора псевдослучайных битов (генератора ПСП). На вход генератора записывается ключ, а на выходе читаются псевдослучайные биты. Длина ключа может составлять от 40 до 2048 бит.

Ядро алгоритма состоит из функции генерации ключевого потока. Другая часть алгоритма – функция инициализации, которая использует ключ переменной длины Ki для создания начального состояния генератора ключевого потока. В основе алгоритма – размер блока или слова, определяемый параметром n. Обычно n = 8, но можно использовать и другие значения. Внутренне состояние шифра определяется массивом слов (S-блоком) размером 2n. При n = 8 элементы блока представляют собой перестановку чисел от 0 до 255, а сама перестановка зависит от ключа переменной длины. Другими элементами внутреннего состояния являются 2 счетчика (каждый размером в одно слово; обозначим их i и j) с нулевыми начальными значениями. В основе вычислений лежит операция по mod 2n.

Генератор ключевого потока RC4 переставляет значения, хранящиеся в S, и каждый раз выбирает различное значение из S в качестве результата. В одном цикле RC4 определяется одно n-битное слово K из ключевого потока, которое в последующем суммируется с исходным текстом для получения зашифрованного текста. Эта часть алгоритма называется генератором ПСП.

Байт K используется в операции XOR с открытым текстом для получения 8-битного шифртекста или для его расшифрования. Так же достаточно проста и инициализация S-блока. Этот алгоритм использует ключ, который подается на вход пользователем. Сначала S-блок заполняется линейно: S0 = 0, S1 = 1, …, S255 = 255. Затем заполняется секретным ключом другой 256-байтный массив. Если необходимо, ключ повторяется многократно, чтобы заполнить весь массив: K0, K1, …, K255. Далее массив S перемешивается путем перестановок, определяемых ключом.

**Практическая часть**

Стартовое окно приложения представлена на рисунке 1.

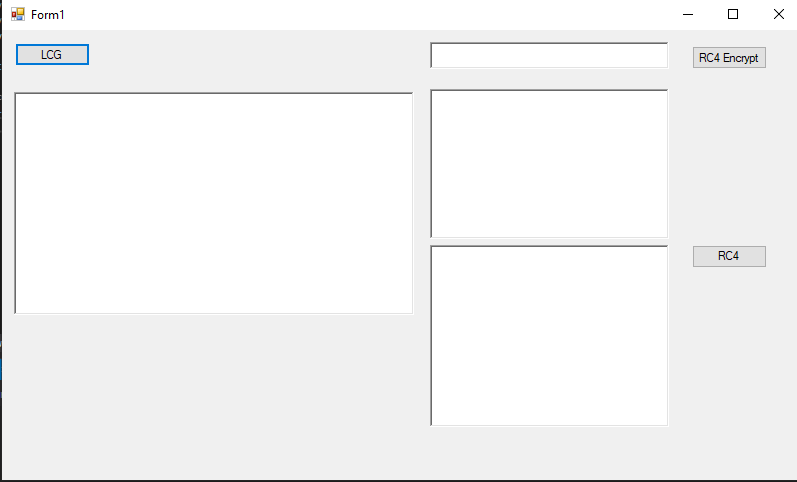


Рисунок 1 – Стартовое окно приложения

При нажатии на кнопку LCG в поле ниже будет выведено первые 100 чисел псевдослучайной последовательности, генерируемой линейным конгруэнтным генератором. Результат выполнения данной операции приведен на рисунке 2.

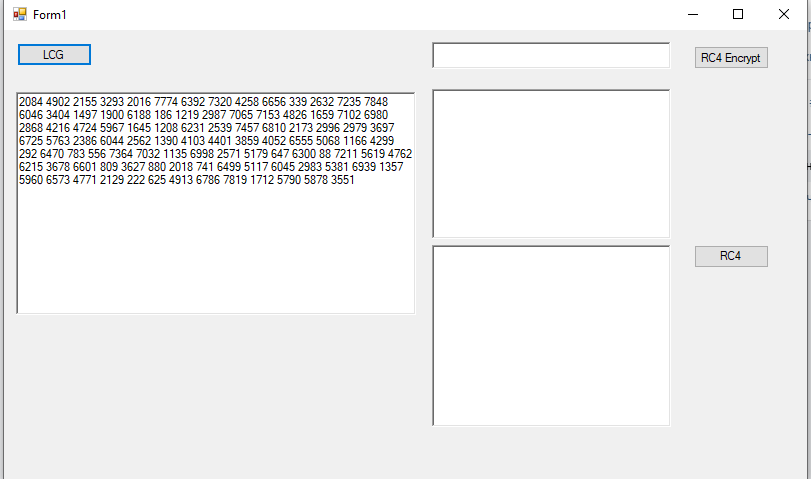


Рисунок 2 – Результат генерации 100 псевдослучайных чисел

При нажатии на кнопку RC4 Encrypt будет произведено зашифрование текста с помощью ключа, результат данной операции представлен на рисунке 3.

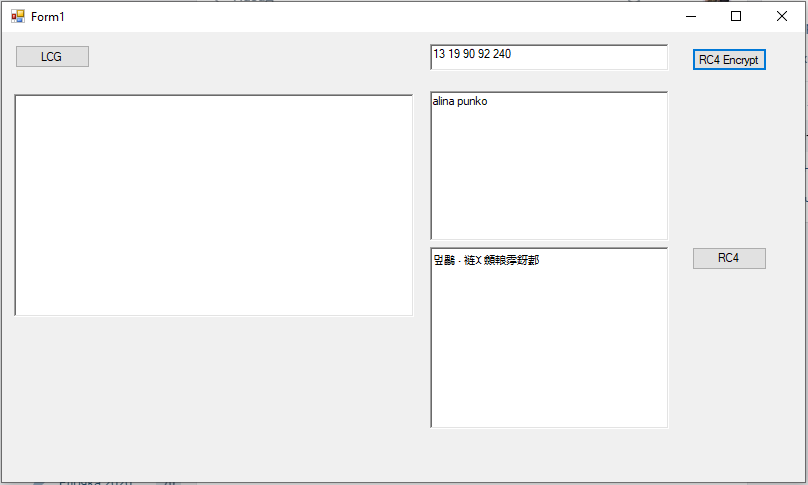


Рисунок 3 – Результат зашифрования

При нажатии на кнопку RC4 Decrypt будет произведено дешифрование текста с помощью ключа, результат данной операции представлен на рисунке 4.

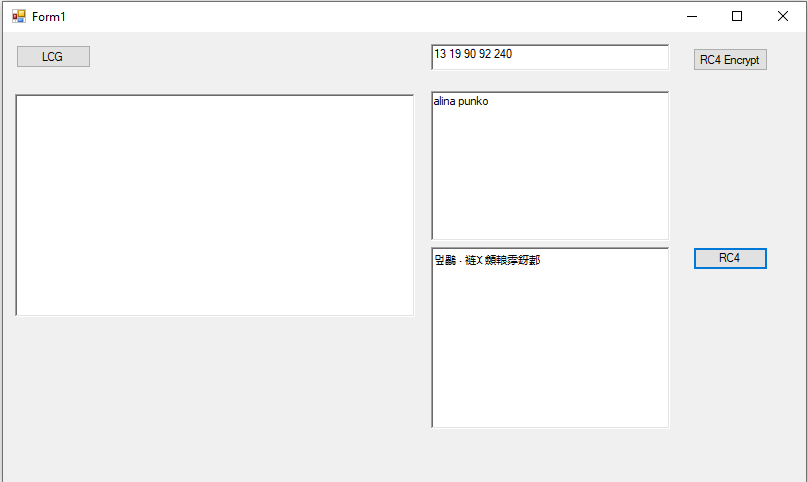


Рисунок 4 – Результат дешифрования

Алгоритм, реализующий генерацию псевдослучайных чисел с использованием линейного конгруэнтного генератора, представлен в листинге 1.

int[] congruential(int x) // функция генерации псевдослучайных чисел

{

int[] result = new int[7875];

int n = 7875, // генерация псевдослучайных чисел в

//диапазоне значений от 0 до 7875 (выбирается случайно n > 0)

a = 421, // множитель (выбирается случайно 0 <= a <= n)

c = 1663; // инкрементирующее значение (выбирается случайно 0 <= c <= mn

for (int i = 0; i < n; i++)

{

x = ((a \* x) + c) % n; // формула линейного конгруэнтного метода генерации псевдослучайных чисел

result[i] = x;

}

return result.Take(100).ToArray();

}

Листинг 1 –Генерация псевдослучайых чисел

Класс, реализующий работу с алгоритмом RC4, представлен в листинге 2. Функция init реализует начальную настройку S-блока.

public class RC4

{

byte[] S = new byte[256];

int x = 0;

int y = 0;

public RC4(byte[] key)

{ init(key);}

private void init(byte[] key)

{

int keyLength = key.Length;

for (int i = 0; i < 256; i++)

{

S[i] = (byte)i;

}

int j = 0;

for (int i = 0; i < 256; i++)

{

j = (j + S[i] + key[i % keyLength]) % 256;

S.Swap(i, j);

}

}

public byte[] Encode(byte[] dataB, int size)

{

byte[] data = dataB.Take(size).ToArray();

byte[] cipher = new byte[data.Length];

for (int m = 0; m < data.Length; m++)

{cipher[m] = (byte)(data[m] ^ keyItem());}

return cipher;

}

public byte[] Decode(byte[] dataB, int size)

{

return Encode(dataB, size);

}

private byte keyItem()

{

x = (x + 1) % 256;

y = (y + S[x]) % 256;

S.Swap(x, y);

return S[(S[x] + S[y]) % 256];

}

**Вывод**

В процессе выполнения данной работы был изучены различные поточные шифры, методы их работы, из достоинства и недостатки. Кроме того было создано приложение, реализующее генерацию псевдослучайных чисел и шифрование/дешифрование текста по алгоритму RC4.